D&D# 08223/0200098-US0

SYSTEM AND METHOD FOR N-DIMENSIONAL ENCRYPTION

Field of the Invention

The present invention relates to data security, and in particular to a method and system for encoding and storing data securely using an n-dimensional entity.

Background of the Invention

A major problem in securing any computing system is determining a safe place to store and hide sensitive information. No matter how many 'lockboxes' are wrapped around this information, one is often still left with the problem of hiding the master key to the outermost lockbox.

It is generally accepted that a safe place to hide this master key is somewhere other than on the system being secured. This is exemplified by the use of smart cards, remote key servers, and the like, for securing the system that contain the master key. However, such approaches remain problematic where there is no remote storage mechanism available.

Storing the master key on the same computing system that it secures, also introduces numerous security concerns. Because, no current hiding place is perfectly secure, anything saved in a persistent store on the same computing system can be located, and hence is vulnerable to analysis, and attack. For example, many computer systems have at least one user that has super-user, or other administrative privileges, which makes many operating system security approaches ineffective for hiding the master key. Moreover, a persistent store and supporting code may be moved from one computing system to another, making it readily available to a hacker. Additionally, a hacker may repeatedly send data to traditional encryption tools that one may employ to secure the master key. In this manner, one may dynamically determine how the master key is secured, making such encryption tools even less effective. Therefore, there is a need in the industry for a method and system for protecting the

master key, and similar sensitive information. Thus, it is with respect to these considerations, and others, that the present invention has been made.

Summary of the Invention

The present invention is directed to addressing the above-mentioned shortcomings, disadvantages and problems, and will be understood by reading and studying the following specification.

The present invention provides a system and method directed to encoding and storing data securely using an n-dimensional entity. The n-dimensional entity is initially populated with bits from a pseudo-random number generator. Plaintext is bitwise translated to a direction and offset from a current cursor position to a bit meeting the "match" criteria with the plaintext bit within the n-dimensional entity. A match may be state driven such that an ON bit in the plaintext may be matched to an OFF bit in the n-dimensional entity, and so forth, depending on the state set by a preceding op-code action. Subsequent cursor directions within the n-dimensional entity are determined using a true random number generator. The resulting output is an encoded array that may be further obfuscated by combining the output with corresponding elements in an obfuscation table.

In one aspect of the invention, a method is directed to encrypting a data string. An n-dimensional entity is generated that comprises random bits. For each bit in the data string, a number of bits are read from the n-dimensional entity. An action is performed that is based in part on the read number of bits. A bit sequence is generated, and a direction within the n-dimensional is selected based in part on the generated bit sequence. An offset is determined between a cursor position and a match bit within the n-dimensional entity. The match bit is based in part on the action, the direction, and the each bit in the data string. The determined offset is then employed to modify the generated bit sequence to generate an encoded data string.

In another aspect of the invention, a method is directed to encrypting a data string. An n-dimensional entity is generated that is populated with pseudo-random bits. For each bit in the data string, a cursor position, and a direction within the n-

dimensional entity is determined. A number of bits are determined by reading bits from the n-dimensional entity from the determined cursor position along the determined direction. An action is performed that is based in part on the determined number of bits. A bit sequence is generated and another direction is selected that is based in part on the bit sequence. An offset is determined between a match bit within the n-dimensional entity and the cursor position. The match bit is based in part on the action, the other direction, and the each bit in the data string. The offset is employed to modify the bit sequence to generate an encoded data string for each bit in the data string.

In still another aspect of the invention, a system is directed to encrypting a data string. The system includes an entity generator and a mapper. The entity generator is arranged to generate an n-dimensional entity. The mapper is arranged to receive the n-dimensional entity and perform actions. The mapper receives the data string. For each bit in the data string, the mapper reads a number of bits from the n-dimensional entity and performs an action based in part on the read number of bits. The mapper also generates a bit sequence for each bit in the data string, and selects a direction within the n-dimensional entity based in part on the generated bit sequence. The mapper further determines an offset between a cursor position and a match bit within the n-dimensional entity. The match bit is based in part on the action, the direction, and the each bit in the data string. Additionally, the mapper modifies the generated bit sequence with the determined offset to generate an encoded data string.

In yet another aspect of the invention, an apparatus is directed to encrypting a data string. The apparatus includes a transceiver and an n-dimensional encrypter. The transceiver receives the data string and sends an encoded array. The n-dimensional encrypter is coupled to the transceiver and is arranged to perform actions. The n-dimensional encrypter generates an n-dimensional entity that is comprised of random bits. For each bit in the received data string, the n-dimensional encrypter reads a number of bits from the n-dimensional entity, and performs an action associated with the read number of bits. The n-dimensional encrypter also generates a bit sequence and selects a direction within the n-dimensional entity based in part on the generated bit sequence, for each bit in the received data string. Moreover, the n-dimensional

encrypter determines an offset between a cursor position and a match bit within the n-dimensional entity. The match bit is based in part on the action, the direction, and the each bit in the received data string. Modifying the generated bit sequence with the determined offset generates an encoded data string. The encoded data string represents a row within the encoded array.

Brief Description of the Drawings

Non-limiting and non-exhaustive embodiments of the present invention are described with reference to the following drawings. In the drawings, like reference numerals refer to like parts throughout the various figures unless otherwise specified.

For a better understanding of the present invention, reference will be made to the following Detailed Description of the Preferred Embodiment, which is to be read in association with the accompanying drawings, wherein:

FIGURE 1 illustrates one embodiment of components of an n-dimensional encryption system;

FIGURE 2 illustrates an exemplary environment in which the present invention may be practiced;

FIGURE 3 illustrates one embodiment of an encoded array;

FIGURE 4 illustrates one embodiment of a table of op-codes and associated actions for use with the n-dimensional entity;

FIGURE 5 illustrates one embodiment of a three-dimensional entity (cube) populated with pseudo-random bits;

FIGURE 6 illustrates a flow diagram generally showing one embodiment for a process of encrypting a data string;

FIGURE 7 illustrates a flow diagram showing one embodiment for a process of generating the n-dimensional entity;

FIGURE 8 illustrates a flow diagram showing one embodiment for a process of employing the n-dimensional entity to encode bits in the data string;

FIGURE 9 illustrates a flow diagram generally showing one embodiment for a process of decrypting an encoded array; and

FIGURE 10 illustrates a flow diagram showing one embodiment for a process of employing the n-dimensional entity to decode bits in the data string, in accordance with the present invention.

Detailed Description of the Preferred Embodiment

The present invention now will be described more fully hereinafter with reference to the accompanying drawings, which form a part hereof, and which show, by way of illustration, specific exemplary embodiments by which the invention may be practiced. This invention may, however, be embodied in many different forms and should not be construed as limited to the embodiments set forth herein; rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art. Among other things, the present invention may be embodied as methods or devices. Accordingly, the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. The following detailed description is, therefore, not to be taken in a limiting sense.

The term "coupled," and "connected," include a direct connection between the things that are connected, or an indirect connection through one or more either passive or active intermediary devices or components.

The terms “comprising,” “including,” “containing,” “having,” and “characterized by,” include an open-ended or inclusive transitional construct and does not exclude additional, unrecited elements, or method steps. For example, a combination that comprises A and B elements, also reads on a combination of A, B, and C elements.

The meaning of "a," "an," and "the" include plural references. The meaning of "in" includes "in" and "on." Additionally, a reference to the singular includes a reference to the plural unless otherwise stated or is inconsistent with the disclosure herein.

Briefly stated, the present invention is directed towards a system and method of encoding and storing data securely using an n-dimensional entity. A user

provides a cursor position within the n-dimensional entity, and a user seed to a pseudo-random number generator. The user seed may be combined with a fingerprint of a computing system in which the invention operates. The n-dimensional entity may be populated with bits from the pseudo-random number generator. Bits within the n-dimensional entity are associated with op-code actions to be performed at each cursor position. Plaintext is then bitwise translated to a direction and an offset from the cursor position to a bit matching the plaintext bit within the n-dimensional entity. The offset is employed to modify a row of truly random bits within an encoded array. Subsequent cursor directions within the n-dimensional entity may be determined using the true random number generator. The output is the encoded array, which may be further obfuscated by, among other way, exclusive or-ing it with corresponding elements in an obfuscation table. The plaintext may be recovered by decoding the encoded array using the original user seed, cursor position within the n-dimensional entity, and fingerprint.

Illustrative Operating Environment

FIGURE 1 illustrates one embodiment of components of an n-dimensional encryption system. System 100 may include many more, or less, components than those shown, however, those shown are sufficient to disclose an illustrative embodiment for practicing the invention.

As shown in the figure, system 100 includes n-dimensional encrypter (NDE) 102, user selectable inputs 104, plaintext 106, and encoded array 108. N-dimensional encrypter 102 includes fingerprinter 110, entity generator 112, cursor normalizer 114, mapper 116, obfuscation table 118, op-code action table 120, random sequence generators 122, and n-dimensional entity 124.

Entity generator 112 is in communication with user selectable inputs 104, cursor normalizer 114, fingerprinter 110, n-dimensional entity 124, and random sequence generators 122. Mapper 116 is in communication with plaintext 106, cursor normalizer 114, random sequence generators 122, op-code action table 120, obfuscation table 118, n-dimensional entity 124, and encoded array 108.

User selectable inputs 104 include, but are not limited to, a user seed, a cursor position within n-dimensional entity 124, a default op-code size, plane direction, and the like. The cursor position may be input in a variety of configurations, including coordinate positions within n-dimensional entity 124. For example, the cursor position may include an X, Y, and Z coordinate position within the n-dimensional entity, where the dimension, n, is three.

The user selectable inputs 104 may also include a starting op-code that is configured to set a portal bit (start bit) within n-dimensional entity 124. The starting op-code may be employed as a first action within n-dimensional entity 124.

User selectable inputs 104 may be provided to NDE 102 employing virtually any mechanism, including a keyboard, mouse, voice activation, touch-screen, another program, and the like.

Plaintext 106 includes virtually any data string that is to be encrypted. In one embodiment, plaintext 106 further includes information as to the length, in bits, of the data string to be encrypted. In another embodiment, the length of plaintext 106 is determined from plaintext 106 by entity generator 112.

Encoded array 108 is described in more detail below in conjunction with FIGURE 3. Briefly, however, encoded array 108 represents the results of encoding plaintext 106 using NDE 102. Encoded array 108 is an M by S array, where M is the number of rows, and S is the number of columns.

Fingerprinter 110 is configured to generate a fingerprint based in part on one or more unique and/or identifiable elements associated with the computing system in which the present invention operates. Examples of such elements include, but are not limited to, a Central Processing Unit's (CPU's) kernel calculated speed, CPU serial number, CPU family identity, CPU manufacturer, an operating system globally unique identifier (GUID), hardware component enumerations, Internet Protocol (IP) address, BIOS serial number, disk serial number, kernel version number, operating system version number, operating system build number, machine name, installed memory characteristic, physical port enumeration, customer supplied ID, MAC address, and the like.

Fingerprinter 110 may convert one or more of the unique and/or identifiable elements into a digest employing a hashing mechanism. Each digest may again be hashed to provide the single fingerprint. Fingerprinter 110 may employ any of a variety of hashing mechanisms including, but not limited to, Message Digests (MD), Secure Hash Algorithms (SHA), and the like. In one embodiment, SHA-1 is employed to generate the fingerprint.

Fingerprinter 110 is further configured to provide the fingerprint to entity generator 112. In one embodiment, entity generator 112 is configured to employ the fingerprint to seed a pseudo-random number generator, such as included within random sequence generators 122.

Random sequence generators 122 include a true random number generator (tRAND), and one or more pseudo-random number generators (pRANDs (1-P)). PRANDs (1-P) may be configured to provide one or more pseudo-random sequences to entity generator 112 for use in the generation of deterministic elements, such as n-dimensional entity 124, and the like. TRAND may be configured to enable a determination of a random cursor movement within n-dimensional entity 124 during an encoding phase. TRAND may also be configured to generate a truly random sequence for use in populating encoded array 108.

TRAND may be configured to provide a sequence of bits that are considered to be statistically truly random. In one embodiment, tRAND is dependent on the computing system in which the present invention operates. In another embodiment, tRAND operates as an entropy pool seeded at a provisioning time, and updated from a time/entropy server. Output of the time/entropy server may include digitally signed and encrypted packets that are received by random sequence generators 122.

In one embodiment, the entropy pool is updated by passing the contents of the current entropy pool and an entropy seed that is read every time an update is invoked. This may be done numerous times and an entropy value is generated from the unsigned number of bits in the entropy pool. The entropy pool is the result of the current entropy pool seed being hashed using any of a variety of hashing mechanisms,

including SHA-1, with the current contents of the entropy pool. Each entropy seed is created by any of a variety of mechanisms. In one embodiment, each entropy seed is created by multiplying an internal CPU counter by an operating system time and securely hashing the results. This is repeated sixteen times.

Entropy sources may include, but are not limited to, disk latency, floating TTL lines at a port, a keystroke delta, and the like. In one embodiment, the system does not include a true entropy source.

PRANDs (1-P) may include virtually any random number generator having an output that is statistically deterministic. PRANDs (1-P) may be implemented using hardware, software, and a combination of hardware and software. In one embodiment, random sequence generators 122 include two pseudo-random number generators, pRAND1, and pRAND2.

PRANDs (1-P) may be seeded independent from each other. PRANDs (1-P) may also be seeded employing values derived in part from fingerprinter 110 in combination with the user seed. By seeding pRANDs (1-P) in this manner the generated series of random numbers may be made unique to the computing system in which the present invention operates and to a particular user. Although such uniqueness is not necessary to the success of the present invention, it does provide another level of obfuscation and a simple mechanism for binding the structure of the present invention to a single computing system.

Op-code action table 120 is described in more detail in conjunction with FIGURE 4. Briefly, however, op-code action table 120 includes an op-code that maps bits within n-dimensional entity 124 to an action to be taken upon its structure. The size and meaning of the op-code are typically implementation specific.

Although, op-code action table 120 may be implemented as a table, it is not so limited. For example, op-code action table 120 may also be implemented as a list, a linked-list, a database, a program, and the like.

Obfuscation table 118 includes an element for each of the possible op-codes within op-code action table 120. Obfuscation table 118 may be populated with random numbers generated through pRANDs (1-P). In one embodiment, pRAND2 is

employed to provide the random numbers. It is noted that obfuscation table 118 is optional and need not be included.

N-dimensional entity 124 represents an n-dimensional array of pseudo-random bits, where N may be any integer value greater than zero. N-dimensional entity 124, typically, does not hold any information about plaintext 106. Rather, n-dimensional entity 124 is configured to enable the generation of encoded array 108, for plaintext 106.

To simplify illustrations, n-dimensional entity 124 is shown as a cube. FIGURE 5 illustrates one embodiment of a three-dimensional entity (cube) populated with pseudo-random bits. As shown in FIGURE 5, bits within the cube may be set to a high value (ON), a low value (OFF), and the like. For example, as shown in the figure, bit 502 is set ON. The present invention however, is not limited to bi-state values for bits within the n-dimensional entity. As such, other levels of states may be employed without departing from the scope of the present invention.

The length (number of bits) of each dimension of n-dimensional entity 124 may be derived in part by employing a random number generated by pRANDs (1-P). In one embodiment, the generated random number is constrained by modulus arithmetic.

Because a very large length for a dimension of n-dimensional entity 124 may adversely affect performance of some slower computing systems, a maximum length may be desired. If a maximum length is desired, it may be determined through any of a variety of engineering approaches that enable tuning of the length for a given computing system. In one embodiment, the length of each dimension is limited to 1K bits.

Moreover, although not required, it may also be desirable to have the overall size of n-dimensional entity 124 to be no larger than a natural size of a CPU integer. Thus, in one embodiment, the size of n-dimensional entity 124 is 32 bits.

N-dimensional entity 124 need not be generated completely at any given time. For example in one embodiment, n-dimensional entity 124 is generated such that bits that are under, or within a defined region, such as around the cursor position, are

generated at any one time. In one embodiment, the cursor position may be converted into a count of the number of pseudo-random numbers that are to be employed to substantially the same view under the cursor position, as there would be had n-dimensional entity 124 been fully generated.

Entity generator 112 is configured to generate at least a portion of n-dimensional entity 124. Entity generator 112 may employ process 700 described below in conjunction with FIGURE 7 to generate n-dimensional entity 124.

Entity generator 112 may employ user selectable inputs 104, and the fingerprint generated by fingerprinter 110 to seed one or more pRANDs (1-P). The output of at least one of pRANDs (1-P) may also be employed by entity generator 112 to determine the dimension and lengths of n-dimensional entity 124. Entity generator 112 may further employ at least one of pRANDs (1-P) to populate n-dimensional entity 124 with pseudo-random bits.

Cursor normalizer 114 is configured to receive user selectable inputs 104, including a cursor position, and ensure that the cursor position resides within the boundaries of n-dimensional entity 124. For example, a cursor position might specify an X-position of 55; however, the X-length for n-dimensional entity 124 may only be 30 bits. Therefore, cursor normalizer 114 normalizes the cursor position to within the 30 bits of length for the example n-dimensional entity. Normalization may be performed employing any of a variety of mechanisms, including truncating the cursor position, mapping the cursor position around a given plane of n-dimensional entity 124, and the like. In one embodiment, a cyclical redundancy check (CRC) is applied to each plane of the cursor position to reduce the number of bits to be normalized. The obtained CRC values for each plane are adjusted employing a circular orbiting algorithm, or the like, until each component of the cursor position is within the boundaries of n-dimensional entity 124. Circular orbiting of n-dimensional entity 124 multiple times may be permissible. For example, normalization may include wrapping the cursor position around n-dimensional entity 124 in an order of dimension, such as X to Y, Y to Z, Z to X, and the like, until the cursor bits to move are exhausted.

Moreover, circular orbiting may include orbiting X dimension twice, then moving to Z, then back to X, then to Y three times, and so forth.

Mapper 116 is configured to generate encoded array 108 from plaintext 106. Mapper 116 may perform actions such as described below in conjunction with FIGURE 8 to generate encoded array 108. Note that preceding actions may also modify actions performed by mapper 116. Hence, it may be possible that an op-code in op-code action table 120 may mean employ an op-code action table other than op-code action table 120 (not shown), and the like. The op-code may also mean that until instructed otherwise, from now on, subtract one from every op-code and execute the resulting associated action.

Mapper 116 may, for example, employ, among other components, op-code action table 120, and n-dimensional entity 124 to bitwise translate plaintext 106 to a direction and an offset from a cursor position to a bit matching the plaintext bit within the n-dimensional entity 124. Mapper 116 may employ the offset to modify each row within encoded array 108. Mapper 116 may further employ obfuscation table 118 to further obfuscate encoded array 108.

Mapper 116 also may be configured to generate plaintext 106 based in part on the decoding of encoded array 108. Mapper 116 may perform actions such as described below in conjunction with FIGURE 10 to generate plaintext 106.

Although entity generator 112, cursor normalizer 114, and mapper 116 are illustrated as three distinct components, the present invention is not so limited. For example, entity generator 112, cursor normalizer 114, and mapper 116 may operate within a single component, several components, any combination of components, and a different arrangement of components, and the like, without departing from the scope of the present invention.

Illustrative Computing Environment

FIGURE 2 shows an exemplary computer 200 that may be included in a system implementing the invention, according to one embodiment of the invention. Computer 200 may include many more components than those shown. The components

shown, however, are sufficient to disclose an illustrative embodiment for practicing the invention.

Computer 200 includes processing unit 212, video display adapter 214, and a mass memory, all in communication with each other via bus 222. The mass memory generally includes RAM 216, ROM 232, and one or more permanent mass storage devices, such as hard disk drive 228, tape drive, optical drive, and/or floppy disk drive. The mass memory stores operating system 220 for controlling the operation of computer 200. Any general-purpose operating system may be employed. Basic input/output system ("BIOS") 218 is also provided for controlling the low-level operation of computer 200.

As illustrated in FIGURE 2, computer 200 also can communicate with the Internet, or some other communications network, via network interface unit 210, which is constructed for use with various communication protocols including the TCP/IP protocol. Network interface unit 210 is sometimes known as a transceiver or transceiving device.

The mass memory as described above illustrates another type of computer-readable media, namely computer storage media. Computer storage media may include volatile, nonvolatile, removable, and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. Examples of computer storage media include RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by a computing device.

In one embodiment, the mass memory stores program code and data for implementing operating system 220. The mass memory may also store additional program code and data for performing the functions of computer 200, and NDE 102. One or more applications 250, NDE 102, and the like, may be loaded into mass memory and run on operating system 220.

Computer 200 may also include an SMTP handler application for transmitting and receiving e-mail, an HTTP handler application for receiving and handing HTTP requests, and an HTTPS handler application for handling secure connections. The HTTPS handler application may initiate communication with an external application in a secure fashion.

Computer 200 also includes input/output interface 224 for communicating with external devices, such as a mouse, keyboard, scanner, or other input devices not shown in FIGURE 2. Likewise, computer 200 may further include additional mass storage facilities such as CD-ROM/DVD-ROM drive 226 and hard disk drive 228. Hard disk drive 228 is utilized by computer 200 to store, among other things, application programs, databases, encoded array 108 of FIGURE 1, NDE 102, and the like.

In one embodiment, n computer systems, substantially similar to computer 200, may share data from computer 200, where each n-dimensional entity provides a seed, starting cursor position, and the like. In this embodiment, n end-users may be required with proper keys before data may be encrypted/decrypted.

FIGURE 3 illustrates one embodiment of an encoded array, such as encoded array 108 of FIGURE 1. Encoded array 300 is an M by S array, where M is the number of rows and S is the number of columns. In one embodiment, M is equal to the length of plaintext 106 of FIGURE 1, and S is equal to the length of an op-code. However, the invention is not so limited. For example, the number of row and/or columns may be extended to further obfuscate encoded array 300.

As shown in FIGURE 3, each row, such as row 308, includes PLANE-MEMBER bits 302, NEW-ENTITY bit 304, and OFFSET bits 306. Such bits may be arranged within row 308 in virtually any sequence, location, and the like.

PLANE-MEMBER bits 302 represent a direction within n-dimensional entity of FIGURE 1. For example, PLANE-MEMBER bits 302 may represent a north, south, east, west, near, far direction, and the like. PLANE-MEMBER bits 302 may also represent an X, Y, Z planar direction, angular direction, and the like.

NEW-ENTITY bit 304 indicates whether the n-dimensional entity is regenerated during a given encoding action for the current row within encoded array 300.

OFFSET bits 306 represent a count of bits between a cursor position within the n-dimensional entity, along the direction indicated by PLANE-MEMBER bits 302, to a bit that matches the current bit within plaintext 106 of FIGURE 1.

FIGURE 4 illustrates one embodiment of a table of op-codes and associated actions for use with the n-dimensional entity. As shown in the figure, table 400 includes op-codes 402 and associated actions 404. Op-codes 402 typically represent a sequence of bits that may be read from the n-dimensional entity. The number of bits read is equal to the size of an op-code within op-codes 402. In one embodiment, the size of an op-code is 16 bits.

Op-codes 402 and their associated actions 404 may be uniquely and randomly generated for each encoding session. The sequence of bits representing an op-code may be generated to be deterministic, non-deterministic, or a combination of deterministic and non-deterministic. Generation of op-codes 402 may be performed using pRANDs (1-P), tRAND, and that like. Moreover, the random number generator (pRAND, tRAND) may be reseeded at various intervals during an encoding session. In one embodiment, the various intervals are selected randomly.

The sequence of bits representing an op-code is employed as an index into table 400, to locate associated action 404. The located associated action 404 may result in a new cursor position, switching bit state to search for within the n-dimensional entity, changing the direction within the n-dimensional entity, and the like. For example, op-code 406 results in the action of "reading the next eight bits along the X direction within the n-dimensional entity." Additionally, the number of bits read is "employed to move the cursor position along the X direction."

Table 400 is merely an example of op-codes and associated actions. It is not intended to illustrate a complete set of all possible op-codes, associated actions, and the like. Other op-codes and associated actions may therefore, be employed without departing from the scope of the present invention.

Moreover, table 400 need not be implemented as a table. For example, table 400 may be implemented as a database, a linked-list, a computer program, in hardware, and the like, without departing from the scope of the present invention.

Illustrative Operation for Replicating Content

The operation of certain aspects of the present invention will now be described with respect to FIGURES 6-10. FIGURE 6 illustrates a flow diagram generally showing one embodiment for a process of encrypting a data string, such as plaintext 106 of FIGURE 1. Process 600 may, for example, operate within NDE 102 of FIGURE 1.

Process 600 begins, after a start block, at block 602 when a cursor position is received. The cursor position provides a starting position within the n-dimensional entity. The number of received coordinates may be employed to indicate the dimensions (N) of the n-dimensional entity. For example, a received cursor position of (10, 123, 99, 666, 0) indicates that the n-dimensional entity has a dimension of five. In one embodiment, another input may be received that specifies the number of dimensions, N.

The process proceeds to block 604 where the plaintext to be encoded is received. Optionally, at block 604, the length of the plaintext, in bits, may also be received. The length of the plaintext may also be determined by any of a variety of mechanisms, such as performing a bit count on the received plaintext. Process 600 continues block 606 where a user seed is received.

Process 600 continues next to block 608 where the n-dimensional entity is generated. Block 608 is described in more detail in conjunction with FIGURE 7. Briefly, however, the received user seed and is employed in part to seed random number generator pRAND1. Output of pRAND1 is then employed to create and populate the n-dimensional entity with pseudo-randomly generated bits.

The process continues to block 610, where the received cursor position is normalized to ensure that the cursor position is within the boundaries of the n-dimensional entity. Normalization of the received cursor position may be achieved by

any of a variety of mechanisms. In one embodiment a circular orbiting algorithm is employed that is similar to a screen display, or printer logic cursor normalization approach, as applied to N planes rather than the traditional two planes.

The present invention however is not limited to normalizing the cursor position to within the boundaries. For example, in one embodiment, the n-dimensional entity may acquire texture on its edges. Hence, in one embodiment, textures may arise by allowing dynamic lengthening of a bit wide plane to accommodate a cursor movement rather than an orbit. In this manner, a determination may navigate the edges of the n-dimensional entity and thereby, such extra-dimensional movement may affect subsequent movement.

The process proceeds next to block 612, where an obfuscation table is generated. The obfuscation table is generated with an element for each of the possible op-codes to be generated. The obfuscation table may be populated with pseudo-random numbers generated from a pseudo-random number generator. The pseudo-random number generator may be pRAND1, or another pseudo-random number generator, pRAND2, or the like. It is noted that employing the obfuscation table is optional and need not be included.

Process 600 continues to block 614, where the plaintext is bitwise encoded using the n-dimensional entity that was generated at block 608. Block 614 is described in more detail below in conjunction with FIGURE 8. Briefly, however, an offset from a cursor position within the n-dimensional entity is determined by performing a search for a match to each bit within the plaintext. The offset and direction of the search are employed to modify a truly random sequence. Each modified truly random sequence is exclusively or-ed with a previously modified truly random sequence to generate the encoded array.

The process continues to block 616 where each exclusively or-ed sequence within the encoded array may be optionally exclusively or-ed with its corresponding element in an obfuscation table. In one embodiment, additional raw bits are added to each sequence in the encoded array to further obfuscate the output. Upon

completion of block 616, the process ends. In one embodiment of the present invention, upon completion of block 616, the n-dimensional entity is destroyed.

FIGURE 7 illustrates a flow diagram showing one embodiment for a process of generating the n-dimensional entity. Process 700 may, for example, operate within entity generator 112 of FIGURE 1.

Process 700, begins, after a start block, at block 702, where a fingerprint is generated from a logical and/or physical context of the computing system in which the present invention operates. One or more unique and/or identifiable elements associated with the computer system are hashed employing any of a variety of hashing mechanisms, including but not limited to SHA-1. The resulting digest(s) may be hashed again to generate a single fingerprint.

The process continues to block 704, where the user seed and fingerprint are combined. In one embodiment, the user seed and fingerprint are added together. The combination is then hashed to generate another digest using any of a variety of hashing mechanisms. The process continues to block 706, where this other digest is employed to seed the pseudo-random number generator, pRAND1.

The process continues to block 708, where a length (the number of bits) is determined for each side of the n-dimensional entity. Each length may be set based on a pseudo-random number that is generated by the pseudo-random number generator. In one embodiment, the pseudo-random number generator employed is pRAND1. In one embodiment, each length is limited to 1K bits per side.

Upon completion of block 708, processing continues to block 710, where the n-dimensional entity is populated with pseudo-random numbers from at least one of the pseudo-random number generators. In one embodiment, pRAND1 is employed to populate the n-dimensional entity. Upon completion of block 708, processing returns to perform other actions.

FIGURE 8 illustrates a flow diagram generally showing one embodiment for a process of employing the n-dimensional entity to encode each bit within the plaintext. Process 800 operates to encode each bit into a sequence of bits, as described

in conjunction with FIGURE 3. Process 800 may operate within mapper 116 of FIGURE 1.

Process 800, begins, after a start block, at block 802 when a starting direction is determined. The starting plane direction may be determined in a variety of approaches, including being provided as a user selectable input, as a default value, and the like.

Processing continues to decision block 804 where a determination is made whether there are more bits within the plaintext to encode. If there are no more bits to encode, processing branches to block 826 where additional obfuscation of the encoded array may be performed. If there are more bits to encode, processing continues to block 806, where the next bit in the plaintext is read.

Processing continues at block 808 where an op-code is determined from the n-dimensional entity. Starting at the current cursor position, bits are read along the current direction from within the n-dimensional entity. The number of bits that are read is equal to the size of the op-code. In one embodiment, the op-code size is 16 bits, however, the op-code size is not so constrained. For example, the op-code size may be a user selectable input. Process 800 continues next to block 810.

At block 810, the read number of bits is employed as an index into an op-code action table to determine an associated action to be performed. Associated actions are described in more detail in conjunction with FIGURE 4. Briefly, however, associated actions may include, but are not limited to, using the read number of bits as a number of bits to move the cursor position along a selected plane, abandoning the current n-dimensional entity and employing another n-dimensional entity, flipping the state of an ON bit in the n-dimensional entity to OFF and the state of an OFF bit to ON, and the like.

Upon performing the associated action, processing continues to block 812, where an op-code size of truly random sequence of bits is generated. The generated truly random sequence is used to populate the row in the encoded array corresponding to the current bit position in the plaintext.

Processing proceeds to block 814 where the PLANE-MEMBER in the current row of the encoded array is employed to determine a direction. Additionally, a NEW-ENTITY bit within the current row of the encoded array is zeroed (set OFF).

Processing continues at block 816, where a search is performed along the direction determined at block 814 until a bit is located within the n-dimensional entity that matches the current plaintext bit.

Some op-code's actions performed at block 810 may reverse the state of the current plaintext bit to search for, reverse the state of the bits in the n-dimensional entity, or the like. However, the search performed at block 816 is configured to accommodate such changes. For example, if the op-code's actions reversed the states of the bits, then if current plaintext bit is set ON, the bit being searched for within the n-dimensional entity is an OFF bit.

Process 800 continues at decision block 818, where a determination is made whether a match has been located. If a match is located, processing branches to block 820; otherwise, processing branches to block 824.

At block 824, if no match is located at block 816, then the current cursor position, or the like, is employed to seed a pseudo-random number generator, such as pRAND3. A new n-dimensional entity may also be generated employing a process substantially similar to process 700 of FIGURE 7. Additionally, the NEW-ENTITY bit in the current row of the encoded array is set high (ON) to indicate that a new n-dimensional entity is generated for this row. Processing loops back to 816 to continue the search for a match to the current plaintext bit.

At block 820, if a match is located at block 816, an offset count is determined by counting the number of bits between the current cursor position and the located matching bit within the n-dimensional entity.

Processing continues at block 822, where the offset count is placed into the current row of the encoded array at location OFFSET by overwriting any bits that may currently reside in that location. Processing branches back to decision block 804, where the above process is substantially repeated until there are no more bits in the plaintext to encode.

At decision block 804, when there are no more bits within the plaintext to be encoded, processing continues to block 826. At block 826, each row in the encoded array may be exclusively or-ed with the previous row in the encoded array. The first row in the encoded array may be exclusively or-ed with the last row of the encoded output.

Additionally, at block 804 each element in the obfuscation table created at block 612 of FIGURE 6 is exclusively or-ed with its corresponding row in the encoded array.

Additional mechanisms may be employed to further obfuscate the encoding of the plaintext at block 804. In one embodiment additional rows of pseudo-random bits are appended to the encoded array. In another embodiment, each row, selection of rows, and the like, is extended in length by adding bits, such as pseudo-random bits. Removing pseudo-random bits may also be performed to shorten each row, selection of rows, and the like. Upon completion of block 826, the n-dimensional entity may be destroyed, and processing returns to perform other actions.

FIGURE 9 illustrates a flow diagram generally showing one embodiment for a process of decrypting an encoded array, such as is generated by process 800 in FIGURE 8. Process 900 may, for example, operate within NDE 102 of FIGURE 1.

Process 900 begins, after a start block, where the same initial cursor position is received that was received at block 602 of FIGURE 6. In addition, at block 902, the size of the op-code that was employed to encode the plaintext may also be received.

Processing continues to block 904, where the encoded array is received that were generated from a substantially similar process to process 600 of FIGURE 6. Processing proceeds to block 906, where the same user seed is received as was provided at block 606 of FIGURE 6.

Processing continues to block 908, which is described in more detail above in conjunction with FIGURE 7. Briefly, at block 908 an n-dimensional entity is generated that is the same as the n-dimensional entity that was generated at block 608 of FIGURE 6.

Process 900 flows next to block 910, where the current cursor position is normalized to within the boundaries of the n-dimensional entity generated at block 908. Normalization may employ substantially the same mechanisms as described above at block 610 of FIGURE 6.

Processing continues to block 912 where an optional un-obfuscation table is generated. The optional un-obfuscation table is generated with an element for each of the output op-codes to be generated. The un-obfuscation table is populated with pseudo-random bits that may be generated from pRAND2, such that the resulting un-obfuscation table is identical to the obfuscation table generated at block 616 of FIGURE 6.

Processing proceeds to block 914, which is described in more detail below at FIGURE 10. Briefly, at block 914, the n-dimensional entity generated at block 908 is employed to decode the encoded array into plaintext. Upon completion of block 914, process 900 ends.

FIGURE 10 illustrates a flow diagram showing one embodiment for a process of employing the n-dimensional entity to decode the encoded array. Process 1000 may, for example, operate within mapper 116 of FIGURE 1.

Process 1000 begins, after a start block, at block 1002 where the encoded array that is to be decoded is received. Processing proceeds next to decision block 1004, where a determination is made whether there are more rows in the encoded array to decode. If there are more rows, processing branches to block 1006: otherwise, processing returns to perform other actions.

At block 1006, the un-obfuscated table is exclusively or-ed with encoded array. The first row in the encoded array is exclusively or-ed with the last row in the encoded array.

Processing continues next to block 1008, where starting at the current cursor position, a number of bits equal to the op-code size are read along the current direction of the n-dimensional entity.

Processing proceeds to block 1010, where the number of bits read are used as an index into the op-code action table to locate an op-code action. The op-code

action is then performed. As in FIGURE 8 at block 810, a variety of actions may be performed, including, but not limited to, creating a new cursor position, changing a bit state to search for from ON to OFF or vice versa, changing a direction to search for a bit, creating a new seed from which to generate a new n-dimensional entity for use in a subsequent action, and the like. Processing continues next to block 1012.

At block 1012, the current row in the encoded array is read to obtain the PLANE-MEMBER. The PLANE-MEMBER is used as the current direction of movement within the n-dimensional entity. Processing continues to block 1014, where the OFFSET is also read from the current row in the encoded array. Processing proceeds to block 1016, where the PLANE-MEMBER, OFFSET and State of the n-dimensional entity are used to fetch the plaintext bit from the n-dimensional entity. The final state of the plaintext bit may be adjusted per the current bit state that may have been set by a preceding op-code action. That is, the plaintext bit fetched from the n-dimensional entity may be used as is or more actually represent the opposite of the bit found for the plaintext. In any event, the final bit is then set at the current location within the plaintext. Upon completion of block 1016, processing loops back to decision block 1004, until there are no more rows to be decoded from the encoded array. Processing then returns to perform other actions.

It will be understood that each block of the flowchart illustration, and combinations of blocks in the flowchart illustration, can be implemented by computer program instructions. These program instructions may be provided to a processor to produce a machine, such that the instructions, which execute on the processor, create means for implementing the actions specified in the flowchart block or blocks. The computer program instructions may be executed by a processor to cause a series of operational steps to be performed by the processor to produce a computer implemented process such that the instructions, which execute on the processor provide steps for implementing the actions specified in the flowchart block or blocks.

Accordingly, blocks of the flowchart illustration support combinations of means for performing the specified actions, combinations of steps for performing the specified actions and program instruction means for performing the specified actions. It

will also be understood that each block of the flowchart illustration, and combinations of blocks in the flowchart illustration, can be implemented by special purpose hardware-based systems which perform the specified actions or steps, or combinations of special purpose hardware and computer instructions.

The above specification, examples, and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.